Farthest Point Seeding for Placement of Streamlines

Abdelkrim Mebarki, Pierre Alliez, Olivier Devillers

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Farthest Point Seeding for Placement of Streamlines

Abdelkrim Mebarki — Pierre Alliez — Olivier Devillers

N° 5524

March 2005

Thème SYM

ISSN 0249-6399 ISRN INRIA/RR--5524--FR+ENG

Farthest Point Seeding for Placement of Streamlines

Abdelkrim Mebarki* , Pierre Alliez† , Olivier Devillers‡

Thème SYM — Systèmes symboliques
Projet GEOMETRICA

Rapport de recherche n° 5524 — March 2005 — 26 pages

Abstract: We propose a novel algorithm for placement of streamlines from two-dimensional steady vector or direction fields. Our method consists of placing one streamline at a time by numerical integration started the furthest away from all previously placed streamlines. Such a farthest point seeding strategy leads to high quality placements by favoring long streamlines, while retaining uniformity with the increasing density. We show in a series of comparative results several improvements over state-of-the-art methods for three important aspects: placement quality, simplicity or efficiency. Robustness as well as efficiency is achieved through the use of a Delaunay triangulation to model the streamlines, address proximity queries and determine the biggest voids by exploiting the empty circle property. Our method handles variable density and extends to multiresolution.

Key-words: Streamline placement, farthest point seeding, Delaunay triangulation, variable density, multiresolution.

* Abdelkrim.Mebarki@sophia.inria.fr
† Pierre.Alliez@sophia.inria.fr
‡ Olivier.Devillers@sophia.inria.fr

Unité de recherche INRIA Sophia Antipolis
2004, route des Lucioles, BP 93, 06902 Sophia Antipolis Cedex (France)
Téléphone : +33 4 92 38 77 77 —Télécopie : +33 4 92 38 77 65

Nouvelle stratégie pour le placement de lignes de courant.

Résumé : Nous proposons un nouvel algorithme de placement de lignes de courant à partir d'un champs de vecteurs ou de direction stationnaire et bidimensionnel. Notre méthode consiste à placer les lignes de courant, l'une après l'autre par intégration numérique en partant du point le plus loin de toutes les lignes de courant déjà placées. Cette stratégie permet d'obtenir des placements de haute qualité en favorisant la création de longues lignes de courant, tout en maintenant une densité uniforme. Nous montrons dans une série de comparaisons les améliorations apportées aux méthodes de l'état de l'art pour la qualité du placement, la simplicité, ou la rapidité. La robustesse et la rapidité sont garanties par l'utilisation de la triangulation de Delaunay pour modéliser les lignes de courant, gérer les requêtes de proximité, et déterminer les plus grands vides en tirant profit de la propriété du cercle vide. Notre méthode peut générer notamment des placements à densités variables, ainsi que des séries de placements en multirésolution.

Mots-clés : Placement de lignes de courant, triangulation de Delaunay, densité variable, multiresolution.

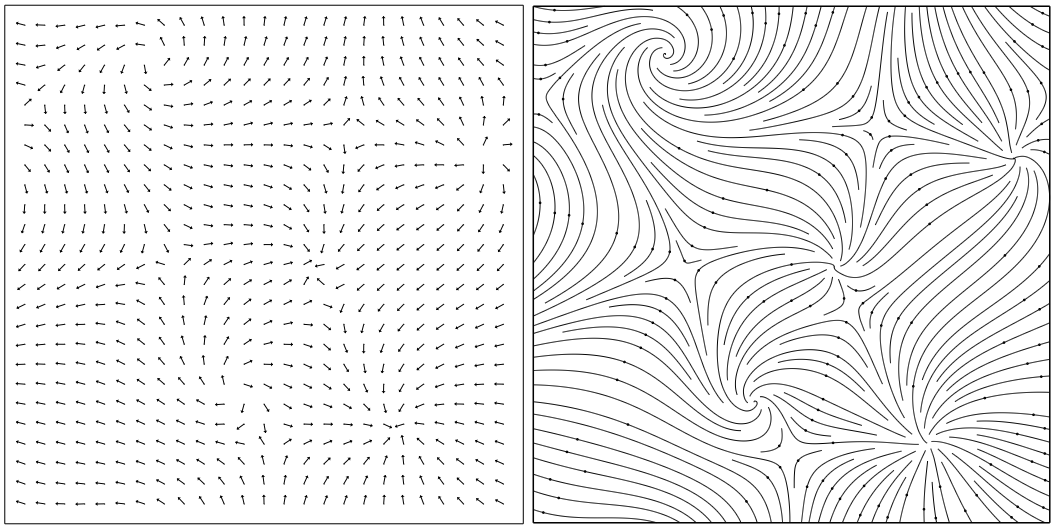


Figure 1: Placement of streamlines with a uniform density(right) according to a given vector field(left). The black dots depict the seed points used for numerical integration. Processing time: 160 ms on a 2GHz Pentium IV.

1 Introduction

Vector and direction fields are commonly used for modeling physical phenomena, where a direction and magnitude, or a vector is assigned to each point inside a domain. A typical example of a vector field is given by the direction, orientation and velocity of a steady wind. An example of a direction field is given by the diffusion in an anisotropic material. In this document a *flow field* refers to either a direction or to a vector field.

Visual depiction of flow fields is motivated by the analysis and exploration of results in scientific computing. One popular method consists of choosing a set of samples throughout the field, and depicting associated *arrow icons* to present a view of the direction, orientation and magnitude in a single picture. The most delicate task of this technique lies into the choice of sample positions so as to best balance between sparse sampling for clear depiction at the risk of missing fine details, versus fine sampling at the risk of a cluttered visualization. For high range vector fields where the magnitude cannot be used to scale the arrow icons, the iconic representation is frequently composed of unit vectors instead, combined with color coding to depict magnitude. Note that other scalar quantities such as flow divergence or curl can be encoded similarly. Other techniques have been developed to obtain a denser depiction by *imaging flow fields*, where the images are obtained by advection of a random noise image [1, 2, 3]. Conversely, some techniques propose to extract only salient features of the flow and depict them as geometric icons to facilitate comprehensive visualization [4].

One of the most popular method in flow visualization consists of placing a set of *streamlines* which are always tangential to the flow in order to emphasis the global field coherency. Another benefit of using such a sparse representation is the possibility to stack a streamline placement on top of an image or any other dense representation. Beside, high quality placement of streamlines has recently proven useful to other applications such as non-photorealistic rendering [5, 6] or curve-based surface remeshing [7, 8]. The versatility of this concept explains our motivation for exploring a new approach to high quality streamline placement (see Fig. 1). We next give a few definitions before giving an overview of existing techniques for high quality placement of streamlines.

Definitions A *streamline* is a curve everywhere tangent to the field. A streamline can be considered as the path traced by an imaginary massless particle dropped into a steady fluid flow described by the field. In practice, a streamline is often represented as a polyline iteratively elongated by bidirectional numerical integration started from a *seed point*, until it comes close to another streamline, hits the domain boundary, reaches a critical point or generates a closed path. A *valid* placement of streamlines consists of saturating the domain with a set of tangential streamlines in accordance with a specified density. A *high quality* streamline placement for visualization has no formal definition, although it is admittedly related to the uniformity of streamlines as well as to the spacing with respect to the desired density. Moreover, long streamlines are preferred to short ones, the goal being to better emphasis the global field temporal coherency. Intuitively, most streamline terminations are perceived as artificial singularities and thus are potential distracters for the observer [9].

2 Related work

When constraining the streamlines to be tangential to the flow, the critical stage of any streamline placement algorithm reduces to the choice of seed points used to start the numerical integration. One trivial solution consists of choosing the seed points on a regular grid, but the resulting streamlines are not evenly spaced and some undesirable patterns frequently appear in the final placement. Another solution is to randomly generate the seed points with a uniform law, but has not shown to improve the placement quality nor to guaranty the domain saturation.

In a pioneering work, Turk and Banks [10] proposed an energy-minimizing approach to generate high quality placements. Their algorithm initially creates a set of so-called *streamlets* (*i.e.*, very short streamlines), then apply a series of energy-decreasing elementary operations to combine, delete, create, lengthen, or shorten the streamlets. To obtain a uniform density of streamlines, the energy to be minimized is related to the difference between a low-pass filtered version of the current placement and a uniform grey image. Although this concept generates high quality placements, the computation time is significant due to the pliant aspect of the algorithm (a method is *pliant* if it incorporates both insertion and deletion, as well as local optimization, see [11]).

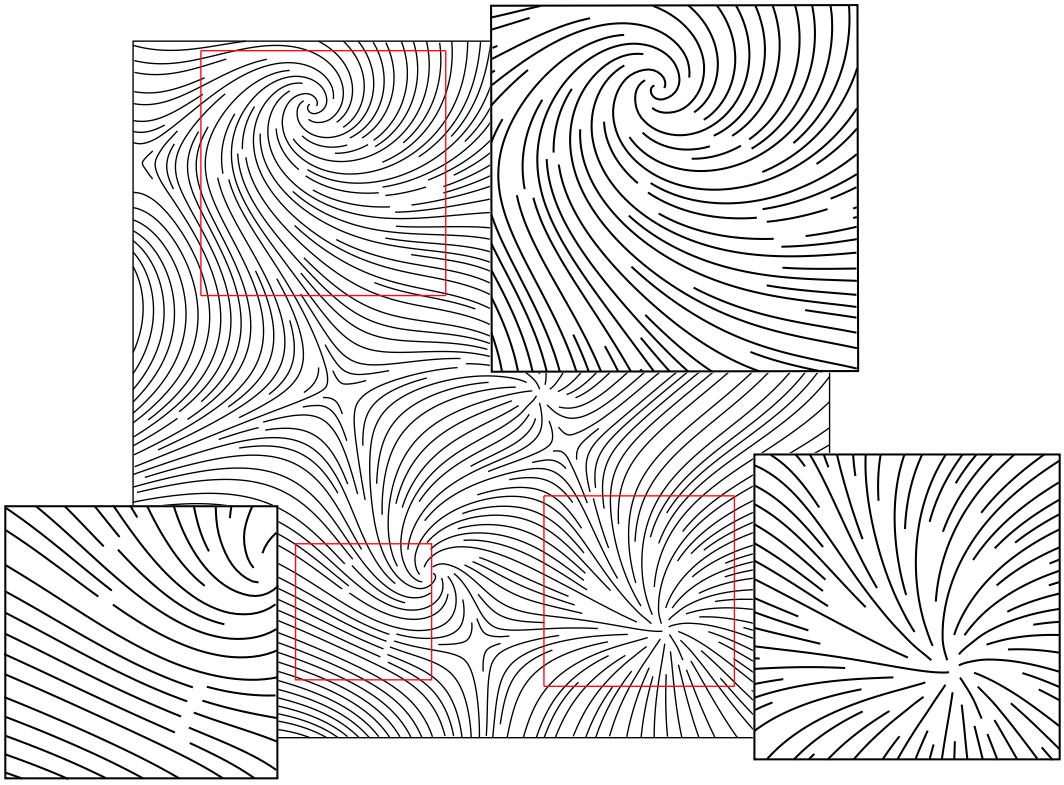
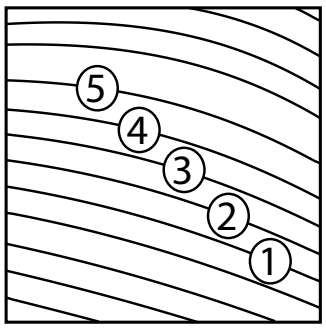


Figure 2: Local seeding strategy produces empty spaces due to the consecutive stopping of a series of streamlines (bottom right closeup). Some discontinuities also appear near singularities (top right closeup) and in laminar areas (bottom left closeup). Figure reproduced from [12].

Jobard and Lefer proposed an algorithm to create evenly spaced streamline placements [12]. Their solution is a greedy placement seeded *in the neighborhood* of previously placed streamlines. During the numerical integration of a streamline, a set of seed points are spread on both sides at a distance slightly higher than the local ideal spacing, and stored into a container of candidate seed points for future placements. The next streamline is placed from a valid seed point randomly chosen and popped out of the container. The algorithm terminates when no more valid seeds remain, *i.e.* when the container is empty. Although providing an excellent balance between efficiency and quality, some empty spaces may remain in the placement, and the length of streamlines is not fully satisfactory. A careful examination of a series of placements produced by this technique led us to characterize the undesirable effects produced by such a local seeding strategy. The first effect is a

large empty space due to the consecutive stopping of a series of streamlines placed sequentially (see Fig. 2). These effects are more frequent when the flow is locally turbulent. Several discontinuities also appear near critical points as well as in homogeneous areas. Another undesirable effect is the concentration of a residual space incident to the last streamline of a sequence when there is not enough room to insert another streamline between the last one in the sequence and a previously placed streamline. Within the local sequence, almost all streamlines are therefore evenly spaced (at least for laminar flows), except for the last one. One desirable effect would be to evenly distribute this residual on a larger area, even at the price of a more global deviation from the local ideal spacing.



To better depict the topology of the flow, Verma *et al.* [13] propose a streamline placement technique where the streamline integration is seeded nearby critical points. In a final step the remaining space is filled by resorting to a random seeding strategy. Although this method provides placements that enhance the flow features and topology, it does not provide precise control over the density. Moreover, seeding streamlines in an area where the flow is frequently highly turbulent does not intrinsically favor long streamlines.

The data structure used to model streamlines as well as empty spaces also plays an essential role in streamline placement algorithms. It is used to accelerate the point location and streamline-to-streamline distance queries. A streamline is frequently approximated by a polyline, each integration step adding a new point at one of its extremities. A common acceleration step consists of approximating the distance queries by simpler point-to-point queries, and the number of associated computations is often reduced by using regular boolean grids combined with region growing procedures [12, 13]. Unfortunately this choice is not suited to high density, and even less suited to variable density. Finally, a streamline placement algorithm which takes only few parameters, runs automatically and produces reproducible results is also valuable.

3 Contributions

The main contribution of our work is a new greedy algorithm which improves over previous work by the quality of placements without compromising the algorithmic simplicity, efficiency, robustness and scalability. To obtain a high quality placement with long and evenly spaced streamlines, our method departs from the usual greedy seeding strategies [12, 13] by choosing the seed points the furthest away from *all* existing streamlines, namely at the center of the biggest voids within the domain. This more global criterion, already used successfully for point sampling and meshing [14, 15, 16] drives the placement procedure by covering the biggest voids in priority. At the intuitive level, seeding a streamline at the farthest point favors *long* streamlines. Our experiments show that the same idea equally distributes the

spaces between streamlines over the domain to obtain an evenly spaced placement, and is amenable to multiresolution placement by retaining the increased density at each newly inserted streamline. Our data structure is a Delaunay triangulation combined with a robust arithmetic from the Computational Geometry Algorithms Library [17]. It has proven both robust and efficient for point location and proximity queries, even for extreme ranges of density specified as input in our experiments. Notice that the Delaunay triangulation not only solves for proximity queries in our algorithm. It also provides us with maximal empty circles, which are candidates for the biggest cavities. Finally, we drastically improve the efficiency of our algorithm by selecting only a small subset of all Delaunay circumcircle centers as candidate seeds for streamline integration. This optimization is shown to be not detrimental to the placement quality.

4 Placement of Streamlines

The *input* of our algorithm is given by (i) a flow *field*, (ii) a *density* specified either globally by the inverse of the ideal spacing distance, or locally by a density field, and (iii) a *saturation* ratio over the desired spacing required to trigger the seeding of a new streamline. The input flow field is given by a discrete set of vectors or directions sampled within a domain, associated with an interpolation scheme (*e.g.* bilinear interpolation over a regular grid) to allow for an evaluation at each point coordinate within the domain. The *output* is a streamline placement, represented as a list of streamlines.

The core idea of our algorithm consists of placing one streamline at a time by numerical integration seeded at the farthest point from all previously placed streamlines. The streamlines are approximated by polylines, whose points are inserted in a 2D Delaunay triangulation. The empty circumscribed circles of the Delaunay triangles provide us with a good approximation of the cavities in the domain. After each streamline integration, all incident triangles which circumcircle diameter is larger (within the saturation ratio) than the desired spacing distance are pushed to a priority queue sorted by the triangle circumcircle diameter. To start each new streamline integration, the triangle with largest circumcircle diameter (and hence the biggest cavity) is popped out of the queue. We first test if it is still a triangle of the triangulation, since it could have been destroyed by a streamline previously added to the triangulation. If not, we pop another triangle out of the queue. If yes, we use the center of its circumcircle as seed point to integrate a new streamline (see Fig. 4). Our algorithm terminates when the priority queue is empty. The size of the biggest cavity being monotonically decreasing, our algorithm guarantees the domain saturation. We summarize our algorithm in the following pseudo-code:

PLACEMENT(*field*,*density*,*saturation*)
Parameters:

- *field* (vector or direction field)
- *density* (inverse of spacing between streamlines)
- *saturation* (ratio over the spacing, > 1)

Variables:

- Visualization_domain *domain*
- Delaunay_triangulation *triangulation*
- list<streamline> *placement*
- priority_queue<triangle> *queue*

Algorithm:

```
begin
    initialization()
    streamline s=PLACE_STREAMLINE(domain.center())
    ADD_STREAMLINE(placement,s)
    while (!queue.empty())
          triangle t=queue.pop()
          if t still in triangulation
            s=PLACE_STREAMLINE(t.circumcenter())
            ADD_STREAMLINE(placement,s)
          endif
    endwhile
    return placement
end
```

initialization() This function inserts in the triangulation a set of points uniformly sampled on a rectangle surrounding the domain boundary. The rectangle is constructed by enlarging the domain boundary with a distance equivalent to the desired separating distance between streamlines (see Fig. 4, 1). This step is required to ensure the domain saturation.

PLACE_STREAMLINE() The streamlines are integrated both forward and backward from a seed point. The integration, performed using first order Euler or second order Runge-Kutta [18], is stopped when the newly integrated point p is located outside the domain, or when the distance between p and the current placement (including part of the current streamline) is smaller than the separating distance. In the next section we explain how we

approximate this distance instead for the sake of efficiency.

```
PLACE_STREAMLINE(seed)

begin
    point p=seed
    streamline s
    do % forward integration %
        s.insert(p)
        triangulation.insert(p)
        p=integrate_forward_from(p)
    while(domain.inside(p) anddistance(p,placement) < 1/density)
    do % backward integration %
        s.insert(p)
        triangulation.insert(p)
        p=integrate_backward_from(p)
    while(domain.inside(p) anddistance(p,placement) < 1/density)
    return s
end
```

4.1 Adding a Streamline

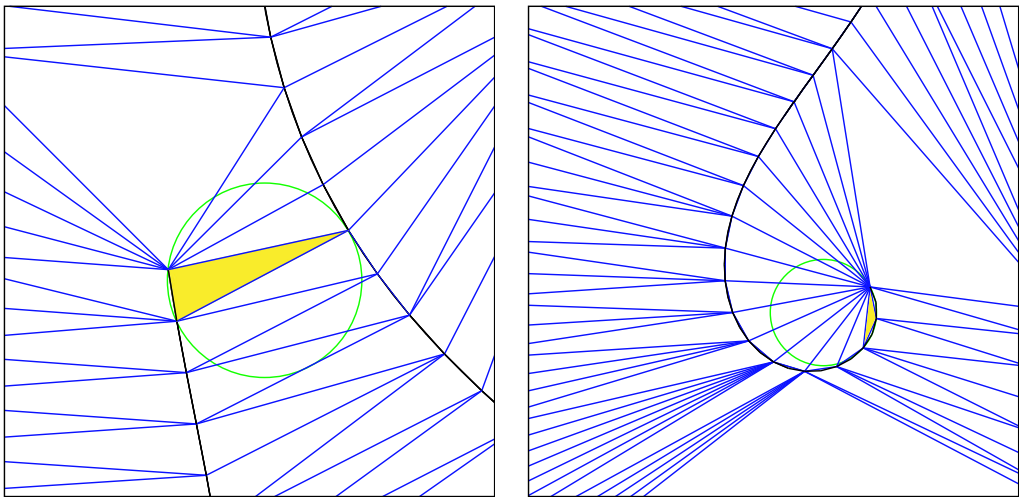
The function ADD_STREAMLINE(*placement*,*streamline*) inserts the newly integrated streamline *streamline* to the placement, and subsequently pushes the set of triangles incident to *streamline* in the priority queue. To reduce the number of triangles pushed in the queue, all triangles which circumcircle diameter is smaller than $saturation \times 1/density$ are not inserted to the queue. Since the saturation ratio is greater than 1 (1.6 used in all our experiments), the latter procedure allows us to avoid the generation of short streamlines when the domain is close to be saturated.

```
ADD_STREAMLINE(placement,streamline)

begin
    placement.insert(streamline)
    for_each(t incident to streamline)
       if (t.circumradius is_local_maximum and t.circumradius > saturation × 1/density)
          queue.insert(t)
       endif
    endwhile
end
```

4.2 Approximating the Distances

When choosing the integration step much smaller than the desired separating distance, a good approximation of the distance between streamlines is given by the minimal distance between the newly integrated point and all other points of the previously integrated streamlines. Notice that when the current streamline comes back to itself or spirals (see Fig.3, b), things become more complicated since the distance computation should also include some – but not all – points of the current streamline (adding all points would always return the integration step as distance). To alleviate this problem and reduce the number of distance computations as well, we insert all integrated points in a Delaunay triangulation, and approximate the distance by the *minimal circumcircle diameter* of the triangles incident to the newly integrated point. The skinny triangles due to the ratio between the integration step and the desired separating distance provide us with a sufficiently accurate approximation of the distance with both regular and spiraling streamlines (see Fig.3).



(a) (b)

Figure 3: The separating distance between streamlines is approximated using the minimal circumdiameter of the triangles incident to the newly integrated point.

4.3 Optimizations

Although the previous section completes the description of our basic algorithm, there is room for improving efficiency. We now describe three optimization steps which are not detrimental to the placement quality.

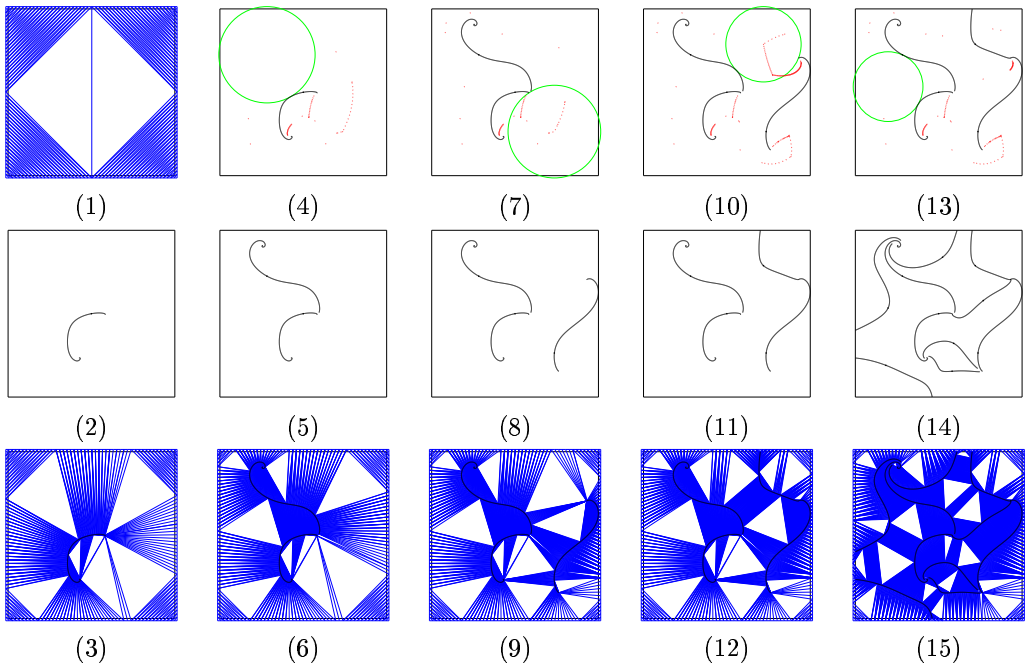


Figure 4: The intermediate stages of our streamline placement algorithm. The final result is shown in Fig.1.

4.3.1 Priority Queue

As mentioned in Section 4.1, adding a streamline to the placement leads to the insertion of all its incident triangles to the priority queue. The priority queue is therefore quickly populated with a large number of triangles. Moreover, a lot of triangles in the queue are in fact short-lived in the triangulation, since placing a new streamline breaks all triangles on its way in the triangulation (see Fig.5).

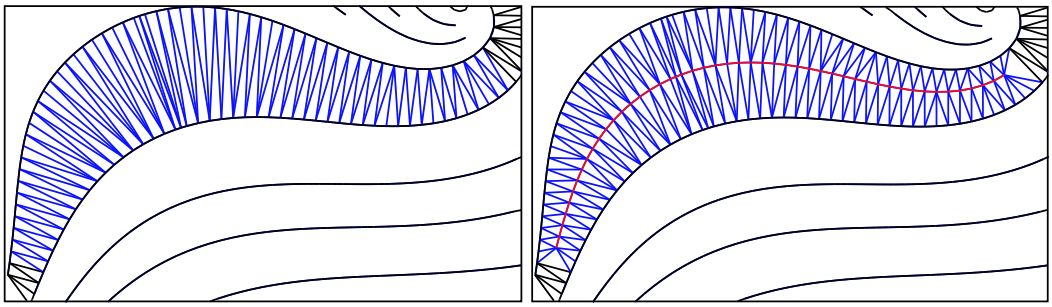


Figure 5: Each newly integrated streamline breaks all triangles on its way.

The vast majority of triangles pushed in the queue will therefore be invalidated when popped out of the queue because they do not belong to the triangulation anymore. To reduce the number of triangles inserted to the priority queue, we walk along the right side of the newly placed streamline, examine the associated sequence of circumcircle radii of all incident triangles, and insert to the queue the triangles which correspond to local maxima. We then apply the same procedure for the left side, and finally insert all triangles incident to the streamline extremities to ensure a proper domain saturation. With almost no impact on the placement quality (see Fig.6), such an optimization on average by 30 the number of triangles pushed in the queue, and therefore produces a significant speed-up of our algorithm (see Table 1).

4.3.2 Sub-Sampling Streamlines in the Triangulation

In our basic algorithm the number of points inserted in the triangulation is equivalent to the number of integrated points. For further optimization it is possible to decouple these numbers by reducing the number of points inserted to the triangulation while keeping a small integration step to generate smooth streamlines. We thus allow the insertion of one point every n integration steps to the triangulation. The parameter n is chosen with respect to the integration step and the desired density so that the approximation of distance queries and the modeling of cavities are accurate enough for our needs. For all examples produced in this paper the ratio n is set to 2.

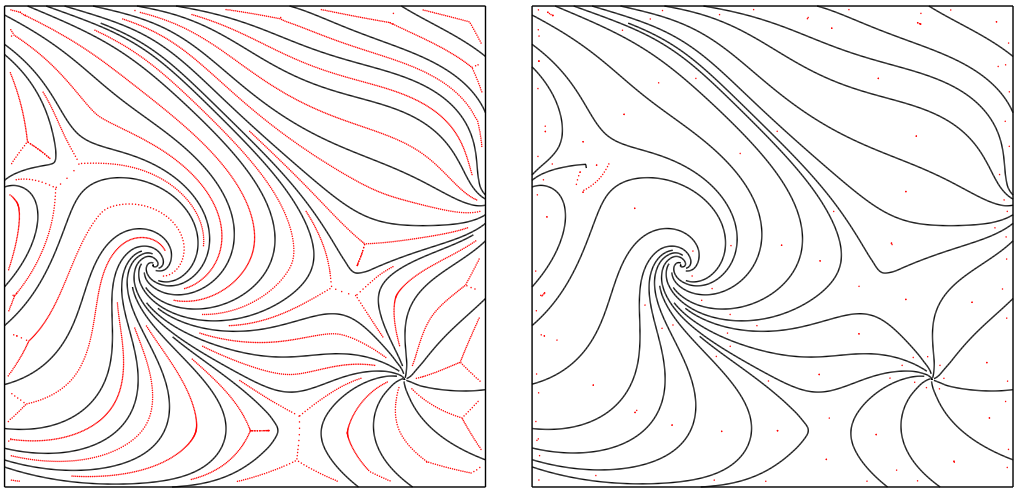
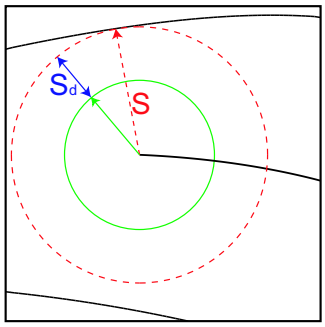


Figure 6: Seed points pushed in the priority queue at an intermediate step of the algorithm, when considering all triangle circumcenters (left), and when choosing only the ones corresponding to local maxima of circumcircle radii (right).

4.3.3 Modifying the Insertion Order

The cost of inserting a point to the Delaunay triangulation is proportional to the number of destroyed triangles. In our case this number is always large because the extremal point of a streamline has often a high degree (see Fig.7,1). To alleviate this issue we delay the insertion of the points to the triangulation after a number m of integration steps, and insert them in *reverse order* of the integration.

This way we pay a high insertion cost once for each m integration steps, and a near-optimal cost for all other steps (see Fig. 7). The parameter m is estimated dynamically during streamline integration as $m = (S - S_d)/dx$, where S stands for the current spacing, S_d for the desired spacing, and dx for the sampling step of points on the triangulation. Notice that such an optimization step does not exempt us from evaluating the distances from all m points to the current streamlines one by one, especially when a variable density is specified. All points which are finally not inserted in the streamline must therefore be inserted in the triangulation to evaluate distances, and removed after. In our experiments the number of such rejections does not exceed 0.5% of the total number of points inserted to



the triangulation. The gain obtained by this reverse insertion order is in practice one order of magnitude higher that the associated over-computations for distance evaluations which occur only at streamline extremities, where the parameter m is often already small.

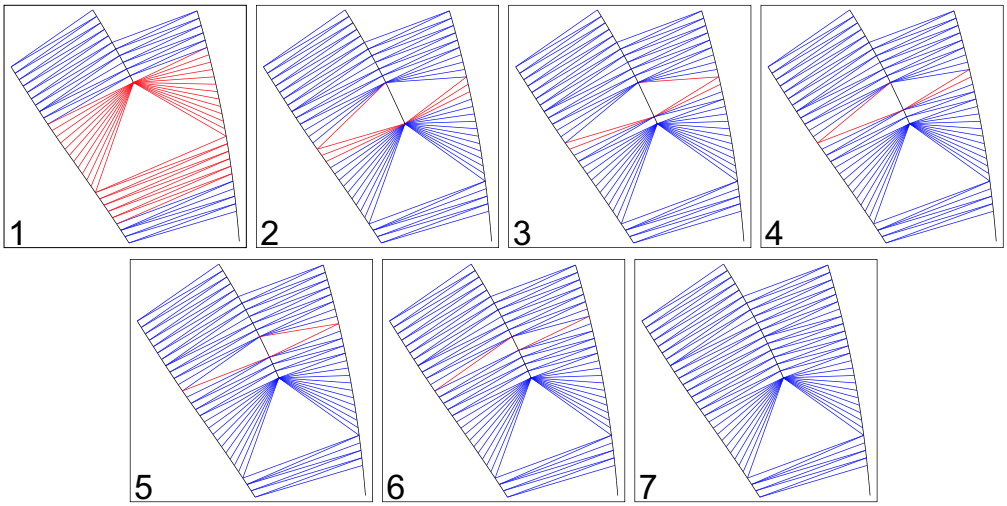


Figure 7: Points are inserted to the triangulation in reverse integration order (the red edges depict the edges conflicted by each insertion step).

Optimization steps	**Duration**
Basic algorithm	620 ms
Optimizing priority queue	550 ms
Sub-sampling lines in the triangulation	220 ms
Modifying insertion order in the triangulation	160 ms

Table 1: Optimization steps and associated durations for the placement shown in Fig. 1.

4.4 Variable density

Our algorithm can take as input a non uniform density field, or simply evaluate any function of the flow such as velocity or vorticity. The desired separating distance simply becomes a function of any point coordinates within the domain (see Fig.8). The triangulation data structure has proven particularly efficient and robust at handling even extreme ranges of density (up to 20K streamlines composed of 8M points in our benchmarks).

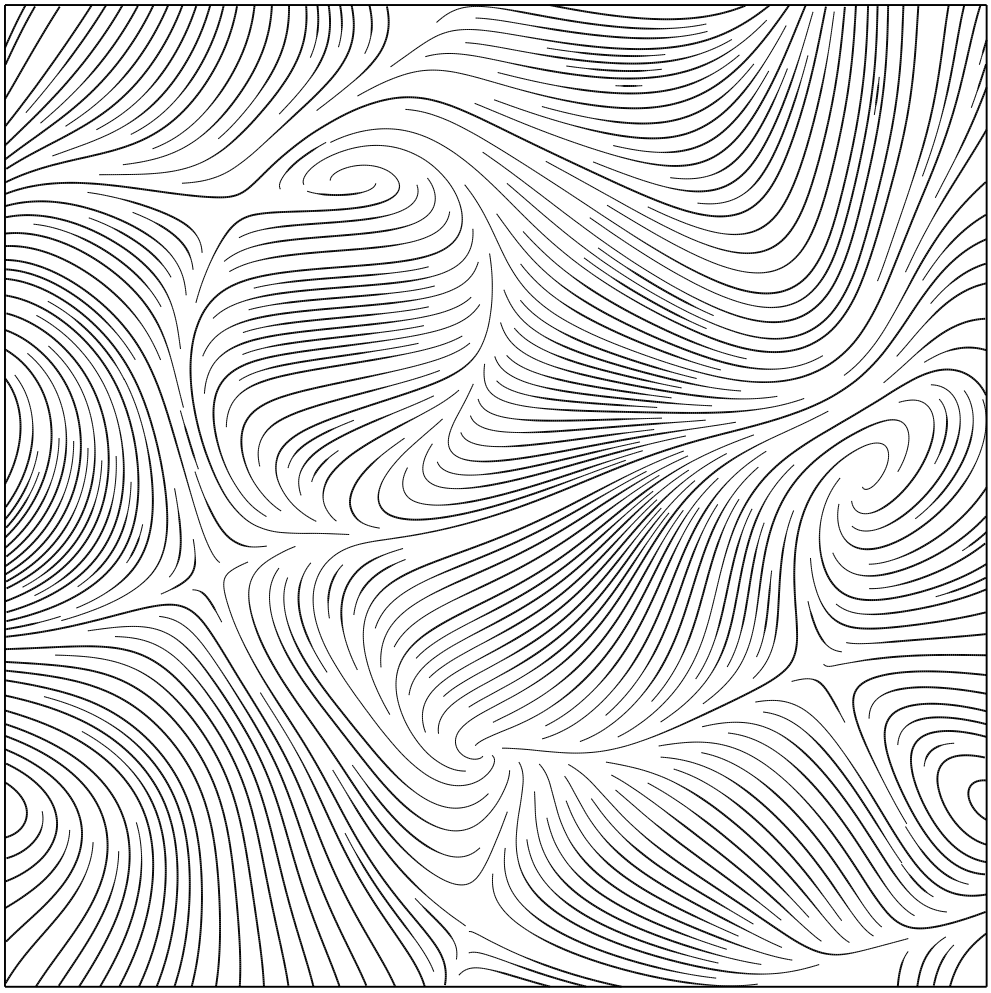


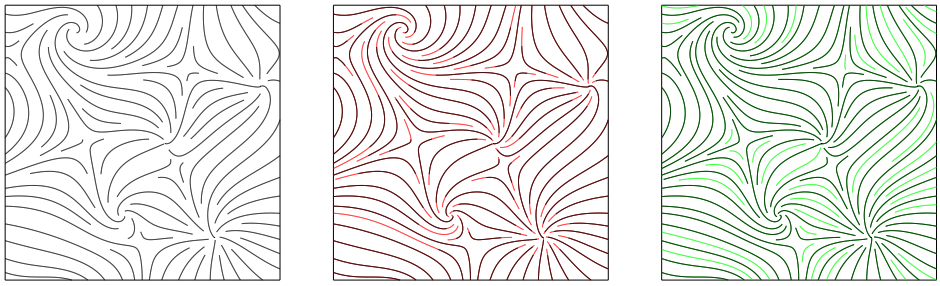
Figure 8: Placement of streamlines with a variable density, in this example related to the flow velocity.

4.5 Multiresolution

Visual exploration of complex flow fields may require multiresolution streamline placement. A typical exploration scenario is a rough depiction of the field on the whole domain, followed by a focused depiction on the regions of interest. One method proposed by Jobard and Lefer [19] performs a sequence of nested streamline placements with increasing density. The streamlines placed at a given level are frozen and used as additional constraints for the finer levels. This approach produces short streamlines added to fill the gaps in-between the streamlines placed at coarser levels. Our method improves over the quality and smoothness of transitions between levels by *elongating all previously placed streamlines* after each increase of the density and before each new placement of streamlines (see Fig.9). The pseudo-code for the multiresolution version of our algorithm is as follows:

```
MULTIRESOLUTION(number_of_levels)

begin
    placement=PLACEMENT(field,density,saturation)
    For_each_level_do
        increase density
        placement.lengthen_streamlines()
        placement.place_new_streamlines()
        return placement
    end_do
end
```



(a) (b) (c)

Figure 9: Multiresolution placement of streamlines: From a given initial low density, the following sequence is applied: (a) place streamlines, increase density, (b) lengthen existing streamlines, (c) place streamlines, etc..

4.6 Implementation

We have implemented our algorithm in C++. Part of the code provided by Greg Turk has been reused to read the vector fields from files as discrete sets of 2D vectors sampled on a regular grid. The 2D Delaunay triangulation and other components for geometric computing are provided by the Computational Geometry Algorithms Library [17]. The priority queue is taken from the Standard Template Library [20]. Overall our implementation takes no more than 2000 lines of code, with 1600 for the placement algorithm itself (remaining part is devoted to file input/output, debugging and display). For comparison, we have also implemented the algorithm from Jobard and Lefer [12] in our framework, using our data structure for efficient evaluation of distance queries. Our algorithm is currently prepared to be submitted as a CGAL package. This package will include an interactive demo running on Linux and Windows.

5 Results

In Fig.10 we depict a series of streamline placements obtained by our algorithm for three types of vector fields and three increasing uniform densities. As shown, our placements saturate the domain with long streamlines, allowing for comprehensive visualization of the flow. The sampling quality is visually pleasing, in particular for turbulent flows. Some undesirable patterns appear for laminar flows, being localized in transition areas between homogeneous regions (see Fig.10,g). The tapering effect introduced by Turk and Banks [10] helps at solving this problem.

One noticeable property of our farthest point seeding strategy is its ability to nicely "wrap" the singularities (see *e.g.*, top left singularity in Fig.1). To better depict this phenomenon and verify that it is specific to our seeding strategy, we have isolated two types of singularity and compared our algorithm with two state-of-the-art methods [10, 12] (see Fig.11). Our algorithm generates a pattern that somehow mimics a multiresolution placement. It is also worthwhile noticing here that our previous attempts at explicitly controlling the placement pattern around singularities by using a seeding strategy similar to the approach from Verma *et al.* [13] produced less pleasing results.

As stated in Section 3, our farthest point seeding strategy favors long streamlines, especially for turbulent flow fields. To better quantify this phenomenon, we measure the length of each newly placed streamline, and compute the difference between cumulated lengths across time (indexed by the number of streamlines) between Jobard-Lefer [12] and our algorithm (see Fig.12). The curve shows that for a given number of streamlines already placed, our algorithm places longer streamlines first (in this example on the first half of the total number of streamlines).

Fig.13 depicts comparative results between Turk-Banks [10], Jobard-Lefer [12] and our algorithm. All pictures from the first two columns are reproduced from the original papers. We now list the noticeable differences for several criteria.

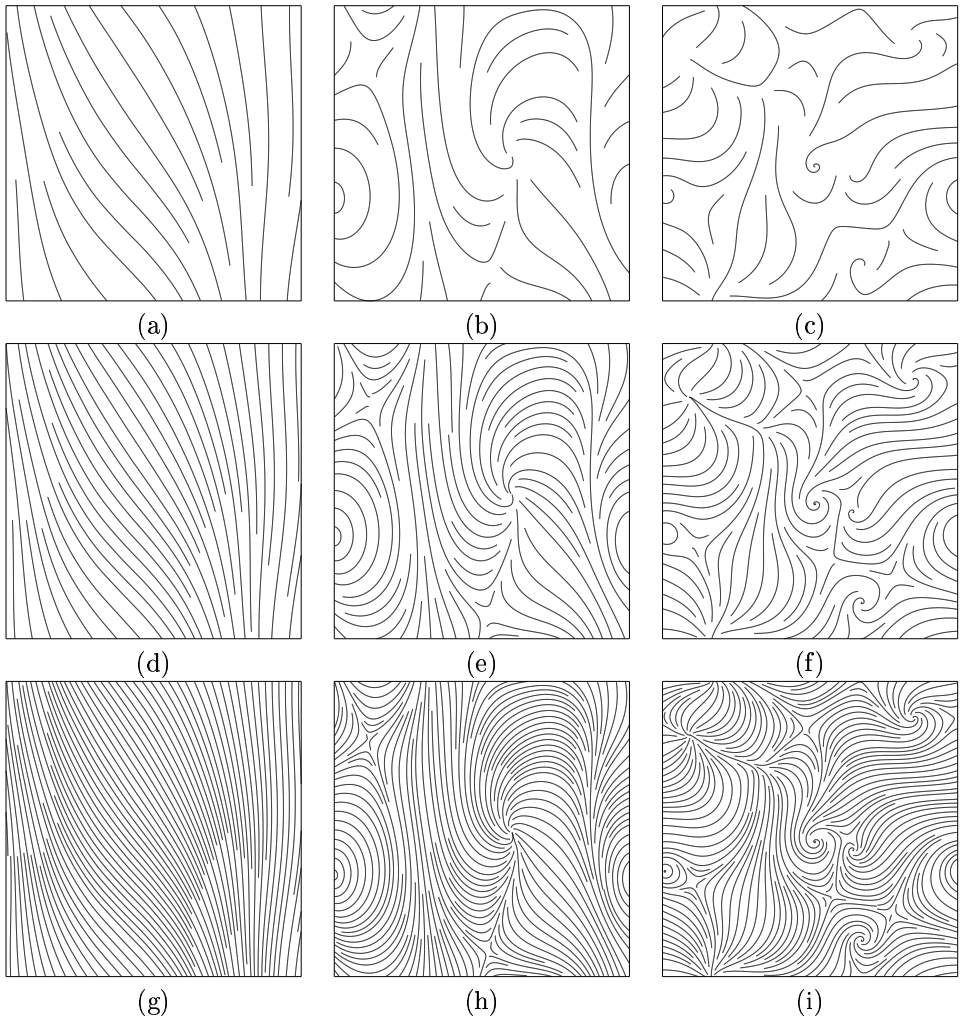
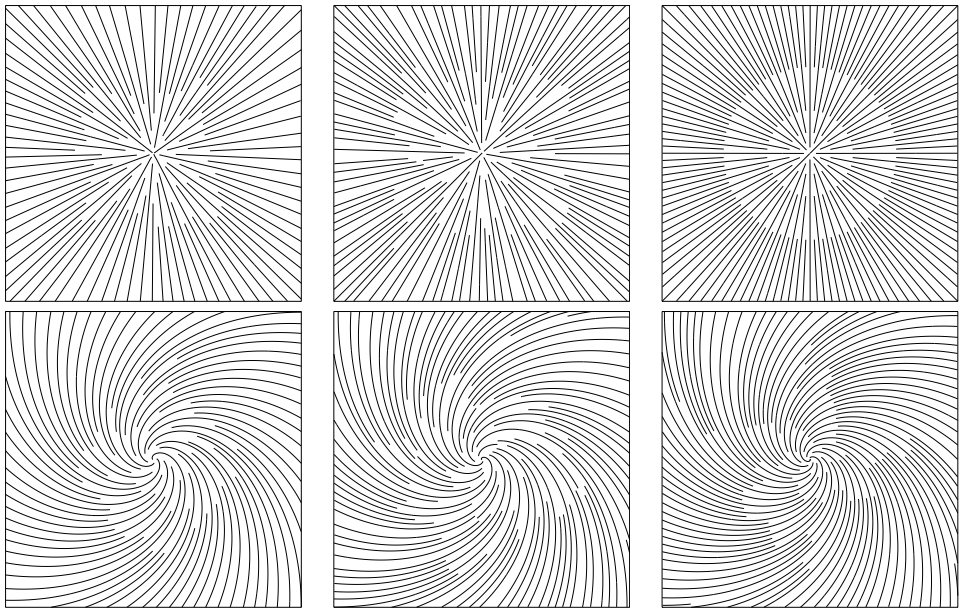


Figure 10: Placement of streamlines on three vector fields, with increasing density (top to down, respectively 4.8, 2.4 and 1.2% requested as separating distance).

Domain saturation Our algorithm saturates the domain by construction. The two other placements also saturate the domain, except for the second column where some free spaces remain, in particular nearby singularities in the densest placement.



Turk-Banks 96 Jobard-Lefer 97 Our algorithm

Figure 11: Comparison of our algorithm with two state-of-the-art techniques [10, 12]. One singularity is isolated per vector field: a source (top) and a spiraling source (bottom).

Flow coherency A high quality placement minimizes the number of discontinuities to better emphasis the flow coherency. Turk-Banks and our algorithm exhibit similar results for both sparse and dense cases. The denser, the more discontinuities appear in the results produced by Jobard-Lefer.

Uniform density Some residual spaces remain in Jobard-Lefer algorithm by placing new seeds at the local ideal spacing distance. Although unavoidable, these spaces are more equally distributed in Turk-Banks and our placements. In essence our algorithm does not intend to match the desired local density perfectly, and produces locally uniform placements instead.

Timings and memory Table 2 summarizes the timings measured on a 2GHz Pentium IV. As already mentioned in Section 4.6 we have implemented the algorithm from Jobard and Lefer [12] to obtain fair measures on a modern computer. For Turk and Banks algorithm, we have executed their software available for download [1]. Our algorithm improves over

[1] http://www.cc.gatech.edu/~turk/streamlines/streamlines.html

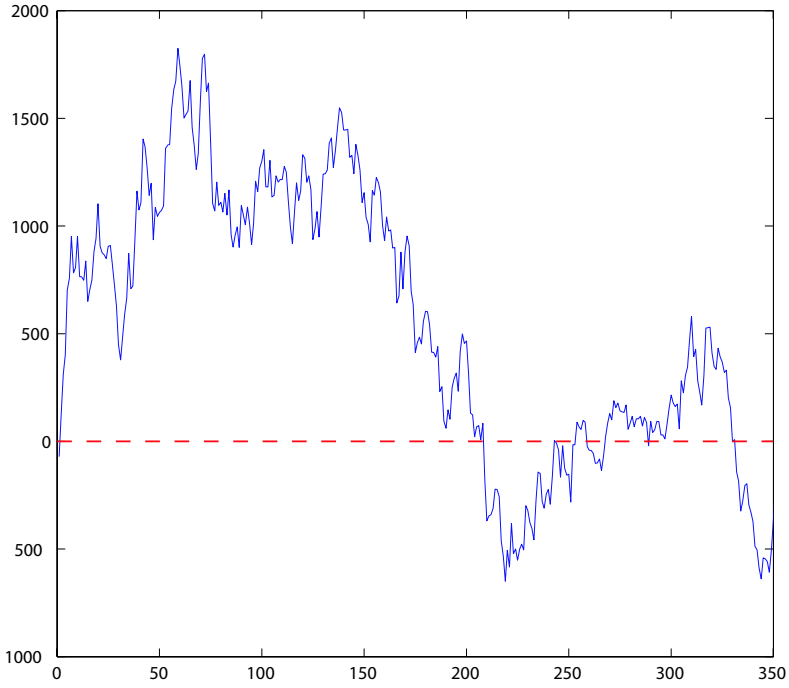


Figure 12: Difference of cumulated lengths (in number of integration steps) across time (indexed by the number of streamlines already placed) between Jobard-Lefer [12] and our algorithm.

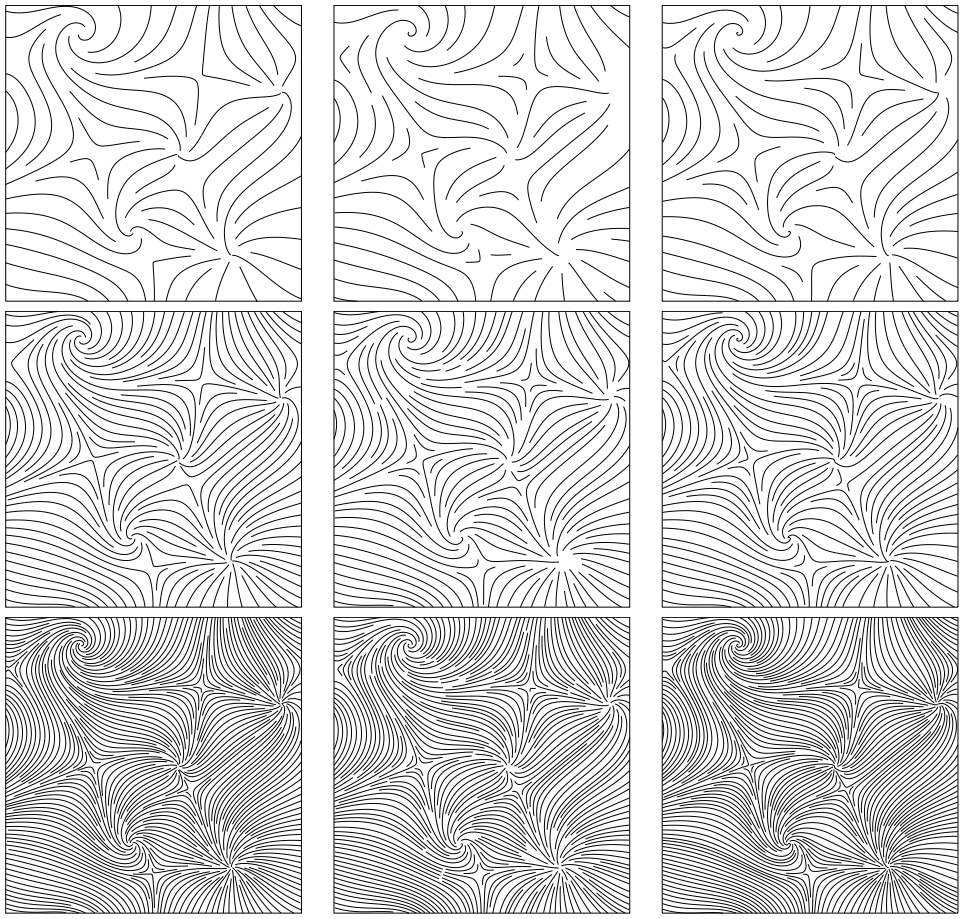


Figure 13: Streamline placements generated by Turk-Banks [10], Jobard-Lefer [12], and by our algorithm for three increasing densities (top to down, 3.36%, 1.68%, 0.84% of the flow width requested as separating distance).

Jobard-Lefer algorithm by the quality of placements, within shorter durations. It produces comparable, and sometimes better results than Turk-Banks algorithm, within significantly shorter durations. Notice that our implementation of Jobard-Lefer algorithm with a Delaunay triangulation data structure is approximately twice as fast as an implementation similar to the original.

Separating dist. (% width)	Our algorithm	Jobard-Lefer	Turk-Banks
0.84	240 ms	790 ms	47 s
1.68	140 ms	290 ms	36 s
3.36	80 ms	130 ms	20 s

Table 2: Timings measured on a 2GHz Pentium IV.

Scalability. We compare the scalability of our algorithm with Jobard-Lefer greedy algorithm. To quantify scalability we measure the duration with respect to the total length of streamlines already placed (expressed in number of integration steps). Our algorithm exhibits a linear behavior (see Fig.14) whereas our implementation of Jobard-Lefer algorithm does not, mainly due to the quickly increasing number of seed points generated (our method uses a sparse priority queue).

6 Conclusion

A novel greedy algorithm for high quality placement of streamlines is proposed. Our algorithm favors the generation of long streamlines by using a farthest point seeding strategy, and ensures the domain saturation by construction. It is simple, efficient, deterministic and involves only two basic algorithmic components which are a priority queue and a Delaunay triangulation both available in widespread libraries [20, 17]. Most functions that trigger the algorithm sequence such as farthest point localizations and distance queries are built upon the Delaunay triangulation. Once parameterized with the appropriate arithmetic, the latter provides us with robustness and scalability.

As future work we plan to add a preliminary step to our algorithm by detecting all closed streamlines as recently proposed by [21], and inserting them to the initial placement. One other possible direction is to explore another variational approach, so as to trade off speed for quality of sampling. Finally, a stimulating challenge would be to extend our algorithm for 3D vector fields as well as for time-varying flows.

Acknowledgment

The authors would like to thank Bruno Jobard for providing us a copy of his Ph.D. thesis and a set of vector fields, Xavier Tricoche for discussion, Greg Turk for the streamline package available on the web, and David Cohen-Steiner for insight about the distance approximation.

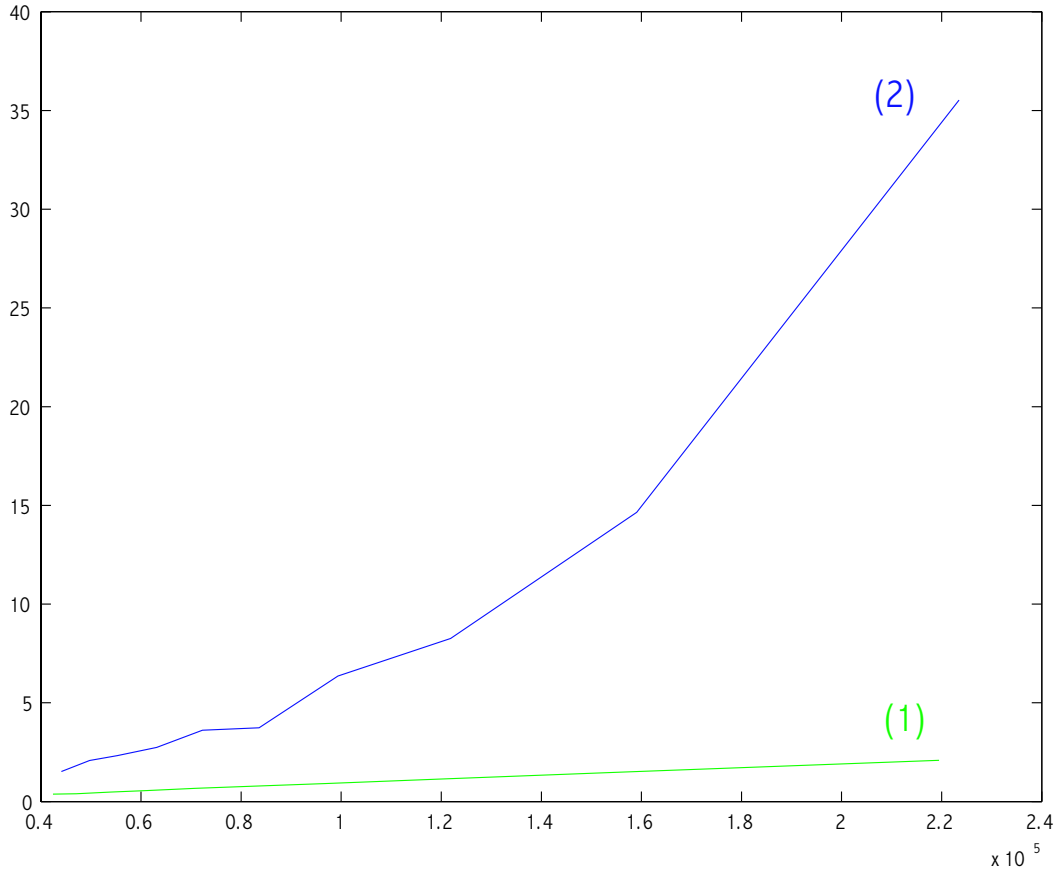


Figure 14: Scalability of our algorithm (green) compared with Jobard-Lefer (blue). We measure the duration (in s) with respect to the total length of streamlines already placed (expressed in number of integration steps).

References

[1] Jarke J. van Wijk. Spot noise texture synthesis for data visualization. In *Proceedings of the 18th annual conference on Computer graphics and interactive techniques*, pages 309–318, 1991.

[2] Brian Cabral and Leith Casey Leedom. Imaging vector fields using line integral convolution. In *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, pages 263–270, 1993.

[3] Robert S. Laramee, Helwig Hauser, Helmut Doleisch Benjamin Vrolijk, Frits H. Post, and Daniel Weiskopf. The state of the art in flow visualization: Dense and texture-based techniques. *Computer Graphics Forum*, 23(2):203–221, 2004.

[4] F. H. Post, B. Vrolijk, H. Hauser, R. S. Laramee, and H. Doleisch. Feature extraction and visualization of flow fields. In *Eurographics 2002 State-of-the-Art Reports*, pages 69–100. European Association for Computer Graphics, The Eurographics Association, 2002.

[5] Christian Rössl and Leif Kobbelt. Line-art rendering of 3d-models. In *Proceedings of the 8th Pacific Conference on Computer Graphics and Applications*, 2000.

[6] Johannes Zander, Tobias Isenberg, Stefan Schlechtweg, and Thomas Strothotte. High Quality Hatching. *Computer Graphics Forum (Proceedings of Eurographics)*, 23(3):421–430, 2004.

[7] Pierre Alliez, David Cohen-Steiner, Olivier Devillers, Bruno Levy, and Mathieu Desbrun. Anisotropic polygonal remeshing. *ACM Transactions on Graphics. Special issue for SIGGRAPH*, pages 485–493, 2003.

[8] Martin Marinov and Leif Kobbelt. Direct anisotropic quad-dominant remeshing. In *Proceeding of Pacific Graphics, Seoul*, 2004.

[9] Bruno Jobard. *Visualisation de champs de vecteurs bidimensionnels à base de streamlines*. PhD thesis, Univ. du Littoral Côte d'Opale, 2000.

[10] Greg Turk and David Banks. Image-Guided Streamline Placement. In *ACM SIGGRAPH Conference Proceedings*, pages 453–460, 1996.

[11] Frank J. Bossen and Paul S. Heckbert. A pliant method for anisotropic mesh generation. In *5th Intl. Meshing Roundtable*, pages 63–74, 1996.

[12] Bruno Jobard and Wilfrid Lefer. Creating Evenly-Spaced Streamlines of Arbitrary Density. In *Proceedings of the Eurographics Workshop on Visualization in Scientific Computing*, pages 45–55, 1997.

[13] Vivek Verma, David T. Kao, and Alex Pang. A Flow-guided Streamline Seeding Strategy. In *IEEE Visualization*, pages 163–170, 2000.

[14] J.-D. Boissonnat and S. Oudot. Provably good surface sampling and approximation. In *Proc. 1st Symp. on Geom. Proc.*, pages 9–18, 2003.

[15] Michael Lindenbaum, Moshe Porat, Yehoshua Y. Zeevi, and Yuval Eldar. The farthest point strategy for progressive image sampling, 1996.

[16] Herbert Edelsbrunner and Damrong Guoy. Sink-insertion for mesh improvement. In *Proceedings of the seventeenth annual symposium on Computational geometry*, pages 115–123, 2001.

[17] CGAL: Computational Geometry Algorithms Library. www.cgal.org.

[18] William H. Press, William T. Vetterling, Saul A. Teukolsky, and Brian P. Flannery. *Numerical Recipes in C++: the art of scientific computing.* Cambridge Univ. press, 2 edition, 2002.

[19] Bruno Jobard and Wilfrid Lefer. Multiresolution flow visualization. In *WSCG 2001 Conference Proceedings*, 2001.

[20] Matthew H. Austern. *Generic Programming and the STL.* Addison-Wesley, 1998.

[21] Holger Theisel, Tino Weinkauf, Hans-Christian Hege, and Hans-Peter Seidel. Grid-independent detection of closed stream lines in 2d vector fields. In *VMV 2004*, 2004.

Contents

Unité de recherche INRIA Sophia Antipolis
2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Unité de recherche INRIA Futurs : Parc Club Orsay Université - ZAC des Vignes
4, rue Jacques Monod - 91893 ORSAY Cedex (France)
Unité de recherche INRIA Lorraine : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399